

# A linear time algorithm to verify strong structural controllability

Alexander Weber, Gunther Reissig and Ferdinand Svaricek

**Abstract**— We prove that strong structural controllability of a pair of structural matrices  $(\mathcal{A}, \mathcal{B})$  can be verified in time linear in  $n + r + \nu$ , where  $\mathcal{A}$  is square,  $n$  and  $r$  denote the number of columns of  $\mathcal{A}$  and  $\mathcal{B}$ , respectively, and  $\nu$  is the number of non-zero entries in  $(\mathcal{A}, \mathcal{B})$ . We also present an algorithm realizing this bound, which depends on a recent, high-level method to verify strong structural controllability and uses sparse matrix data structures. Linear time complexity is actually achieved by separately storing both the structural matrix  $(\mathcal{A}, \mathcal{B})$  and its transpose, linking the two data structures through a third one, and a novel, efficient scheme to update all the data during the computations. We illustrate the performance of our algorithm using systems of various sizes and sparsity.

## I. INTRODUCTION

Strong structural controllability of the pair  $(\mathcal{A}, \mathcal{B})$  of structural matrices  $\mathcal{A} \in \{0, *\}^{n \times n}$ ,  $\mathcal{B} \in \{0, *\}^{n \times r}$  is, by definition, equivalent to the linear system

$$\dot{x}(t) = Ax(t) + Bu(t) \quad (1)$$

being controllable for *all* matrices  $A$  and  $B$  whose positions of the non-zero entries (zero entries) coincide with the positions of the *\*-entries* (0-entries) of  $\mathcal{A}$  and  $\mathcal{B}$ , respectively. Here,  $A$  and  $B$  denote matrices with real or complex entries having the same dimension as  $\mathcal{A}$  and  $\mathcal{B}$ , respectively,  $x$  denotes the real or complex valued  $n$ -dimensional state of (1) and  $u$  is a real or complex valued  $r$ -dimensional input signal. The system given by (1) is controllable if for any initial state and any terminal state, there exists an input signal  $u$  steering the system from the initial to the terminal state [1].

Strong structural controllability of linear time-invariant systems has been extensively studied [2]–[5]. Algorithms to test strong structural controllability of a pair  $(\mathcal{A}, \mathcal{B})$  have been presented in [3] and [5] having complexity  $\mathcal{O}(n^3)$  and  $\mathcal{O}(n^2)$ , respectively. In [6], an algorithm was presented without an analysis of its complexity.

Recently, the notion of strong structural controllability has been extended to linear time-varying systems and characterizations in terms of the zero-nonzero pattern  $(\mathcal{A}, \mathcal{B})$  have been established [6]–[9]. While the conditions differ, it turns out that their verification for a time-varying system can be reduced to the verification of strong structural controllability for an auxiliary time-invariant system (1). This implies that algorithms originally derived to test the strong structural controllability of time-invariant systems may be also used for the time-varying case.

This is the accepted version of a paper published in *Proc. 53rd IEEE Conf. Decision and Control*, 2014.

This work has been supported by the German Research Foundation (DFG) under grant no. RE 1249/3-2. The authors are with the University of the Federal Armed Forces Munich, Dept. Aerospace Eng., Inst. of Control Eng. (LRT-15), D-85577 Neubiberg (Munich), Germany.

In this paper, we prove that strong structural controllability can be verified in time linear in  $n + r + \nu$ , where  $\nu$  is the number of non-zero entries in  $(\mathcal{A}, \mathcal{B})$ . We also present an algorithm realizing this bound, which depends on the recent, high-level method from [6], [9] and uses sparse matrix data structures. Linear time complexity is actually achieved by separately storing both the structural matrix  $(\mathcal{A}, \mathcal{B})$  and its transpose, linking the two data structures through a third one, and a novel, efficient scheme to update all the data during the computations.

The need for fast algorithms becomes evident by the following application of strong structural controllability. The dynamical evolution of complex networks, such as power grids or gene regulatory networks, is commonly studied in terms of linear systems of the form (1), where the entries of  $x$  denote the state of the nodes,  $A$  denotes the adjacency matrix of the underlying graph and  $B$  identifies the nodes that can be controlled from outside the network; see e.g. [10] and the references therein. In real applications, the entries of the matrix  $A$  are not exactly known, which is why one considers its zero-nonzero structure, encoded in the structural matrices  $\mathcal{A}$  and  $\mathcal{B}$ , instead. The particular interest with regard to controllability of networks is then to find a structural matrix  $\mathcal{B}$  with the minimum number of columns such that the given network is strong structurally controllable [5], [11]. This problem was proved to be NP-hard [5]. One way to avoid NP-hardness is to consider the special case in which  $\mathcal{B}$  is required to have precisely one *\**-entry per column, which results in  $\mathcal{O}(n^3)$  time-complexity [11]. Another alternative is to pose the problem in the framework of the so-called weak structural controllability [10], [12], [13]. However, both alternatives suffer from severe drawbacks. Firstly, restricting  $\mathcal{B}$  to some special structure may result in a minimum number of columns that is strictly greater than the number of columns actually required using arbitrary  $\mathcal{B}$ . (An example is given in the present paper.) With regard to economizing the computational effort for input signals that solution is inappropriate. Secondly, the approach based on weak structural controllability yields results that are correct for all pairs of matrices  $(A, B)$  of structure  $(\mathcal{A}, \mathcal{B})$  with the possible exception of a set of measure zero. The possible exceptions may very well be a problem, in particular, when the parameters of the system (1) slowly change over time, so that the submanifold of exceptional points may be passed over with certainty. Therefore, there is much interest to tackle the original NP-hard problem based on strong rather than weak controllability, and fast algorithms are in demand.

The remainder of this paper is organized as follows. Having introduced some notation and terminology in Section II, in Section III we briefly review the method presented

in [6], [9]. Section IV contains the main result about the time complexity for verifying strong structural controllability and an implementable algorithm of such a test. In Section V, several computational results on the performance of an implementation on various structural matrices are presented.

## II. NOTATION AND TERMINOLOGY

The set  $\{1, 2, 3, \dots\}$  of natural numbers we denote by  $\mathbb{N}$ , the set of real and complex numbers by  $\mathbb{R}$  and  $\mathbb{C}$ , respectively, and  $\mathbb{F}$  denotes either  $\mathbb{R}$  and  $\mathbb{C}$ . For  $a, b \in \mathbb{N}$ ,  $a \leq b$ , we write  $[a; b]$  and  $[a; b[$  for the set  $\{a, a+1, \dots, b\}$  and  $\{a, a+1, \dots, b-1\}$ , respectively. For the  $i$ -th entry of  $y \in \mathbb{N}^m$  we write  $y(i)$  ( $1 \leq i \leq m$ ). Moreover, we write  $y \in [a; b]^m$  if  $y(i) \in [a; b]$  for all  $i$ .

$\mathcal{X}$  stands for a *structural matrix*, i.e.  $\mathcal{X} \in \{0, *\}^{n \times m}$ . We say that a matrix  $X \in \mathbb{F}^{n \times m}$  has the *non-zero structure* of  $\mathcal{X}$  if  $X_{i,j} \neq 0$  is equivalent to  $\mathcal{X}_{i,j} = *$  for any  $i, j$ . Here and subsequently,  $X_{i,j}$  ( $\mathcal{X}_{i,j}$ , respectively) denotes the entry in the  $i$ -th row and  $j$ -th column of  $X$  ( $\mathcal{X}$ , respectively).  $\mathcal{A}$  and  $\mathcal{B}$  denote structural matrices of dimension  $n \times n$  and  $n \times r$ , respectively. The transpose of  $\mathcal{X}$  is denoted by  $\mathcal{X}^T$ . For a structural matrix  $\mathcal{X}$  we introduce the following sets. For  $j \in [1; m]$  we define

$$\text{NZR}_{\mathcal{X}}(j) := \{i \in [1; n] \mid \mathcal{X}_{i,j} = *\}.$$

The above set indicates the rows of  $\mathcal{X}$  that have a  $*$ -entry in the  $j$ -th column. For reviewing the results in [6] as outlined in the introduction, we define for a set  $V \subseteq [1; n]$  the set

$$\text{NZC}_{\mathcal{X}}(V) := \{j \in [1; m] \mid \exists i \in V : \mathcal{X}_{i,j} = *\}.$$

Throughout the paper, however, we will omit the subscript  $\mathcal{X}$  as it will be obvious from the context to which matrix the sets are related.

## III. REVIEW OF THE METHOD TO BE IMPLEMENTED

In this section, we state the method for testing strong structural controllability as given in [6] for which we will give an implementable algorithm in the subsequent section. The test consists of computing the set  $V$  as specified in Fig. 1 for both  $L = 0$  and  $L = 1$ . (We adopted the formulation of the test as presented in [9].) For convenience of the reader, we will briefly indicate the role of the two runs by stating the theorem which implies the correctness of the method.

**III.1 Definition.** *The pair  $(\mathcal{A}, \mathcal{B})$  of structural matrices is strong structurally controllable for  $\lambda \in \mathbb{C}$  if the matrix  $(\lambda \text{id} - \mathcal{A}, \mathcal{B})$  has full rank for all pairs of matrices  $(A, B) \in \mathbb{F}^{n \times (n+r)}$  that have the non-zero structure of  $(\mathcal{A}, \mathcal{B})$ . Here,  $\text{id}$  denotes the  $n \times n$  identity matrix.*

A consequence of the well-known Hautus criterion (e.g. [1, Lemma 3.3.7]) is that the pair  $(\mathcal{A}, \mathcal{B})$  is strong structurally controllable if and only if it is strong structurally controllable for all  $\lambda \in \mathbb{C}$ . Based on this fact, the following theorem has been proved in [2].

**III.2 Theorem.** *Consider the following conditions for the structural matrix  $\mathcal{X} = (\mathcal{A}, \mathcal{B})$ :*

**Input**  $L, (\mathcal{A}, \mathcal{B})$

**Require:**  $L \in \{0, 1\}$

```

1:  $V := [1; n]$ 
2: while  $V \neq \emptyset$  do
3:   if  $L = 0$  then
4:      $T := \{v \in [1; n+r] \mid |V \cap \text{NZR}(v)| = 1\}$ 
5:   else
6:      $T := \{v \in [1; n+r] \setminus V \mid |V \cap \text{NZR}(v)| = 1\}$ 
7:   end if
8:   if  $L = 0$  or  $V \subseteq \text{NZC}(V)$  then
9:     if  $T = \emptyset$  then
10:      break
11:    end if
12:    Pick  $v \in T$ .
13:     $\{w\} := \text{NZR}(v)$ 
14:  else
15:    Pick  $w \in V \setminus \text{NZC}(V)$ .
16:  end if
17:   $V := V \setminus \{w\}$ 
18: end while

```

**Output**  $V$

Fig. 1. Method to test if  $(\mathcal{A}, \mathcal{B})$  is strong structurally controllable [6].

( $G_0$ ) For every non-empty subset  $V \subseteq [1; n]$  of row indices of  $\mathcal{X}$  there exists a column index  $v \in [1; n+r]$  such that  $V \cap \text{NZR}(v)$  is a singleton,

( $G_1$ ) For every non-empty subset  $V \subseteq [1; n]$  of row indices of  $\mathcal{X}$  that satisfies  $V \subseteq \text{NZC}(V)$  there exists  $v \in [1; n+r] \setminus V$  such that  $V \cap \text{NZR}(v)$  is a singleton.

Condition ( $G_0$ ) holds if and only if  $\mathcal{X}$  is strong structurally controllable for  $\lambda = 0$ . Analogously, condition ( $G_1$ ) holds if and only if  $\mathcal{X}$  is strong structurally controllable for every  $\lambda \in \mathbb{C} \setminus \{0\}$ . In particular,  $\mathcal{X}$  is strong structurally controllable if and only if both ( $G_0$ ) and ( $G_1$ ) hold.

The proof of the above theorem as given in [6] proves the following theorem.

**III.3 Theorem.** *The pair  $(\mathcal{A}, \mathcal{B})$  is strong structurally controllable*

(i) *for  $\lambda = 0$  if and only if the algorithm in Fig. 1 returns the empty set for  $L = 0$ ,*

(ii) *for every  $\lambda \neq 0$  if and only if the algorithm in Fig. 1 returns the empty set for  $L = 1$ .*

*In particular,  $(\mathcal{A}, \mathcal{B})$  is strong structurally controllable if and only if both runs of the algorithm return the empty set.*

It is important to note that although conditions ( $G_0$ ) and ( $G_1$ ) require verifications for every non-empty subset  $V \subseteq [1; n]$ , Theorem III.3 implies that a test of merely  $n$  such subsets is sufficient. Nevertheless, a brute-force implementation of Fig. 1 will not lead to a linear time test since the computation of the sets  $T$  and  $\text{NZC}(V)$  is complex.

In the following section, we present an algorithm that realizes the method given in Fig. 1 in linear time. The key to linear time complexity is combining sophisticated data structures and sparse matrix techniques.

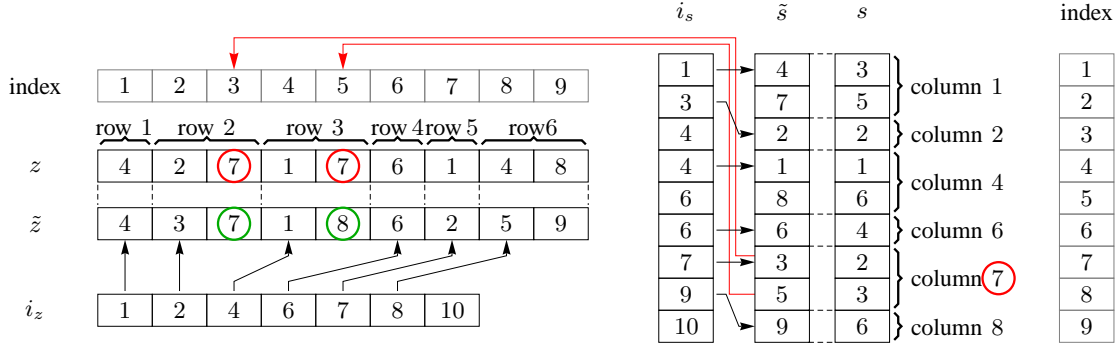


Fig. 2. Data structures for  $(\mathcal{A}, \mathcal{B})$  in Example IV.2. The usage of the array  $\tilde{s}$  is indicated: The positions in  $z$  whose entry is column 7 are stored in positions 7 and 8 of  $\tilde{s}$ . 7 and 8 are the indices of column 7. The entries of  $\tilde{z}$  indicated by the green circles are those that need to be swapped in Example IV.6.

#### IV. THE MAIN RESULT

Our main result, which claims the existence of a linear time test for strong structural controllability, is given in Section IV-A. For its proof we give a particular algorithm for Fig. 1. Specifically, in Section IV-B we discuss the used data structures and the algorithm is presented in Section IV-C.

##### A. Main result

**IV.1 Theorem.** *Let the pair of structural matrices  $(\mathcal{A}, \mathcal{B}) \in \{0, *\}^{n \times (n+r)}$  have  $\nu \in [0; n(n+r)]$  \*-entries. Strong structural controllability of  $(\mathcal{A}, \mathcal{B})$  can be verified with time complexity  $\mathcal{O}(n+r+\nu)$ .*

The following Sections IV-B and IV-C are devoted to the proof of Theorem IV.1. Let us abbreviate the pair of structural matrices  $(\mathcal{A}, \mathcal{B})$  by  $\mathcal{X}$ , and let  $n$ ,  $r$  and  $\nu$  be as in the statement of Theorem IV.1. Without loss of generality, let  $\nu > 0$ .

##### B. Data structures

To obtain linear complexity in the algorithm that we present, we introduce the following sophisticated data structures. To begin with, we will store the matrices  $\mathcal{X}$  and  $\mathcal{X}^T$  separately. The format that we use is well-known in the framework of sparse matrices [14]–[16]. To provide efficient access between the data, we introduce a third, novel data structure which links those of  $\mathcal{X}$  and  $\mathcal{X}^T$ . Moreover, we introduce appropriate data structures for the sets  $T$  and  $V$  as defined in Fig. 1.

1) *Data structure for  $\mathcal{X}$* : The structural matrix  $\mathcal{X}$  is assumed to be available in the compressed column storage format (CCS) [14]. The CCS-format exists in two versions, namely for ordinary matrices and for structural matrices. The latter, suitable for our purposes, consists of two integer arrays  $s$  and  $i_s$  of length  $\nu$  and  $n+r+1$ , respectively. These arrays are defined as follows:

- $i_s(j) - 1$  equals the number \*-entries in the first  $j - 1$  columns of  $\mathcal{X}$ , and
- $\mathcal{X}_{i,j} = *$  if and only if there exists  $k \in [i_s(j); i_s(j+1)]$  such that  $s(k) = i$ .

Note that  $s \in [1; n]^\nu$  and  $i_s \in [1; \nu+1]^{n+r+1}$ . (See Section II for notation.)

##### IV.2 Example. Consider the structural matrices

$$\mathcal{A} = \begin{pmatrix} 0 & 0 & 0 & * & 0 & 0 \\ 0 & * & 0 & 0 & 0 & 0 \\ * & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & * \\ * & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & * & 0 & 0 \end{pmatrix} \quad \text{and} \quad \mathcal{B} = \begin{pmatrix} 0 & 0 \\ * & 0 \\ * & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & * \end{pmatrix}. \quad (2)$$

The \*-entries in column 1 of  $(\mathcal{A}, \mathcal{B})$  are in the rows 3 and 5, hence  $(s(1), s(2))$  may equal  $(3, 5)$  or  $(5, 3)$ . We emphasize that both choices are consistent with our definition.  $s(3)$  equals 2 since the \*-entry in column 2 appears in row 2. For the array  $i_s$  we have  $i_s(1) = 1$  as the first row index related to column 1 is stored in position 1 of  $s$ .  $i_s(2) = 3$  since the first row index related to column 2 is stored in position 3 of  $s$ . The subsequent entries of  $s$  and  $i_s$  are obtained similarly. See also Fig. 2.  $\square$

2) *Data structure for  $\mathcal{X}^T$* : We store  $\mathcal{X}^T$  in its CCS-format and we denote the corresponding arrays by  $z$  and  $i_z$ . Note that  $z \in [1; n+r]^\nu$  and  $i_z \in [1; \nu+1]^{n+1}$ . This data structure is also known as the compressed row storage format of  $\mathcal{X}$  [14].

For simplicity of notation we introduce the following definition.

**IV.3 Definition.** *Let  $i \in [1; n]$  and  $j \in [1; n+r]$ . We say that  $k \in [1; \nu]$  is an index of the row  $i$  if  $k \in [i_z(i); i_z(i+1)]$ , and that  $l \in [1; \nu]$  is an index of the column  $j$  if  $l \in [i_s(j); i_s(j+1)]$ .*

3) *Data structures for linking the data structures of  $\mathcal{X}$  and  $\mathcal{X}^T$* : In the algorithm that we present, we will take advantage of the non-uniqueness of the array  $s$  as follows. Entries in  $s$  will be swapped during the execution of the algorithm in order to store additional information in the ordering of the entries of  $s$  (without violating the properties of  $s$  as a part of the CCS-format of  $\mathcal{X}$ ). The input of a swapping operation will be a row index  $w$ , and the first step will be to identify in constant time the positions  $l$  in  $s$  such that  $s(l) = w$ . The arrays  $z$ ,  $i_z$  provide the column indices  $j$  such that  $\mathcal{X}_{w,j} = *$ , which is the set  $\{z(l) \mid l \in [i_z(w); i_z(w+1)]\}$ . However,  $z$  and  $i_z$  do not provide the positions in  $s$  of the entry  $w$ .

In order to avoid a search operation, we introduce an integer array  $\tilde{z}$  of length  $\nu$  as follows. We define  $\tilde{z}$  such that  $\tilde{z}(l)$  equals the position in  $s$  in which  $w$  is stored among the row indices of the column  $j = z(l)$ .

Analogously, we will introduce an array  $\tilde{s}$  to store the positions of the column indices in the array  $z$ . The array  $\tilde{s}$  will be required to update  $\tilde{z}$  as a swap in  $s$  will require an update of  $\tilde{z}$ .

Before we define  $\tilde{s}$  and  $\tilde{z}$  formally, we identify some entries of  $\tilde{z}$  for the pair  $(\mathcal{A}, \mathcal{B})$  as given in (2).

**IV.4 Example.** We consider the arrays  $s$  and  $z$  as given in Fig. 2. Row 2 of  $(\mathcal{A}, \mathcal{B})$  has  $*$ -entries in columns  $2 = z(2)$  and  $7 = z(3)$ . Among the row indices of column 2 in  $s$ , row 2 appears in position 3, hence we set  $\tilde{z}(2) = 3$ . As for column 7, row 2 appears in position 7 in  $s$ , hence  $\tilde{z}(3) := 7$ .

Let us suppose that we had defined  $z(2) = 7$  and  $z(3) = 2$ , so that the pair  $(z, i_z)$  would still represent the pair  $(\mathcal{A}, \mathcal{B})$  given in (2). In this case, we need to define  $\tilde{z}(2)$  to equal 7 since we require  $\tilde{z}(2)$  to be an index pointing to row indices of column  $z(2)$ . Similarly, in this case,  $\tilde{z}(3) = 2$  since 2 is an index of column  $z(3)$ .  $\square$

The integer arrays  $\tilde{s} \in [1; \nu]^\nu$ ,  $\tilde{z} \in [1; \nu]^\nu$  are formally defined by the following properties:

$$\tilde{s}(k) \text{ is an index of the row } s(k) \text{ for all } k, \text{ and} \quad (3a)$$

$$z(\tilde{s}(l)) = j \text{ if } l \text{ is an index of the column } j, \quad (3b)$$

and similarly,

$$\tilde{z}(l) \text{ is an index of the column } z(l) \text{ for all } l, \text{ and} \quad (4a)$$

$$s(\tilde{z}(k)) = i \text{ if } k \text{ is an index of the row } i. \quad (4b)$$

For later purposes, we show the following lemma which may be used for an alternative definition of  $\tilde{s}$  and  $\tilde{z}$ . It also shows the uniqueness of  $\tilde{s}$  and  $\tilde{z}$  for given  $s$  and  $z$ .

**IV.5 Lemma.** Let  $\tilde{s}_1 \in [1; \nu]^\nu$  satisfy (3a) in place of  $\tilde{s}$ , let  $\tilde{z}_1 \in [1; \nu]^\nu$  satisfy (4a) in place of  $\tilde{z}$ . Then

$$\tilde{z}_1(\tilde{s}_1(k)) = k, \text{ and} \quad (5a)$$

$$\tilde{s}_1(\tilde{z}_1(k)) = k \quad (5b)$$

for any  $k \in [1; \nu]$  if and only if  $\tilde{s}_1$  and  $\tilde{z}_1$  satisfy (3b) and (4b) in place of  $\tilde{s}$  and  $\tilde{z}$ , respectively.

*Proof:* We show (3b) for the array  $\tilde{s}_1$ . The proof of (4b) for  $\tilde{z}_1$  is similar. We first remark that if  $l \in [1; \nu]$  is an index of both the columns  $j$  and  $j_0$  then  $j = j_0$ . This follows immediately from Definition IV.3 and the definition of  $i_s$ .

Let  $l$  be an index of the column  $j$ , hence  $\tilde{z}_1(\tilde{s}_1(l))$  is an index of the column  $j$ . By (4a),  $\tilde{z}_1(\tilde{s}_1(l))$  is an index of the column  $z(\tilde{s}_1(l))$ . On account of the above remark, we have  $z(\tilde{s}_1(l)) = j$ .

Conversely, let  $k \in [1; \nu]$  be an index of the column  $j$ . By (3a) and (4b) we have

$$s(\tilde{z}_1(\tilde{s}_1(k))) = s(k). \quad (6)$$

Since both  $\tilde{z}_1(\tilde{s}_1(k))$  and  $k$  are indices of the column  $j$  by (3b) and (4a), it follows from (6) and the definition of  $s$  that  $\tilde{z}_1(\tilde{s}_1(k)) = k$ . The proof of (5b) is similar.  $\square$

4) *Data structures for sets:* Realizing the method given in Fig. 1 requires the computation of the set  $T$  in lines 4 and 6. Computing  $T$  will require accessing the sets  $V \cap \text{NZR}(v)$  for all  $v \in [1; n+r]$ . For  $L = 1$ , we additionally need to access the set

$$T_0 := \{v \in V \mid V \cap \text{NZR}(v) = \emptyset\}$$

since the test  $V \subseteq \text{NZC}(V)$  in line 8 is equivalent to the test  $T_0 = \emptyset$ .

Therefore, we introduce below appropriate data structures to store the sets  $V \cap \text{NZR}(\cdot)$ ,  $T$ ,  $T_0$  and  $V$ .

To represent the set  $V \cap \text{NZR}(v)$  for any  $v \in [1; n+r]$ , we first note that

$$\text{NZR}(v) = \{s(k) \mid k \in [i_s(v); i_s(v+1)[\}.$$

Therefore, we take advantage of the non-uniqueness of  $s$  as a part of the CCS-format of  $\mathcal{X}$ . In particular, we introduce an integer array  $c$  of length  $n+r$ , and add the following property to the definition of  $s$ :

$$V \cap \text{NZR}(v) = \{s(k) \mid k \in [i_s(v); i_s(v) + c(v)[\}. \quad (7)$$

In other words,  $c(v)$  equals the value  $|V \cap \text{NZR}(v)|$  and indicates the last position in  $s$  of an element of  $V \cap \text{NZR}(v)$ . We remark that if  $V = [1; n]$  then  $c(v)$  equals the number of  $*$ -entries in the  $v$ -th column of  $\mathcal{X}$ .

The pair  $(s, c)$  stores all information about the sets  $V \cap \text{NZR}(\cdot)$ . Moreover, the procedure to remove an element  $w$  from the set  $V \cap \text{NZR}(v)$  can be easily performed: The entry  $w$  and the entry in position  $i_s(v) + c(v) - 1$  are swapped in  $s$ , and  $c(v)$  is decremented by 1.

The data structures for  $T$  and  $T_0$  are such that adding, removing and picking of elements can be achieved in constant time. These requirements can be realized using two integer arrays (two for each set) with appropriate functionality. The set  $V$  is implemented as a boolean array of length  $n+r$  where a '1' in the  $k$ -entry indicates that  $k \in V$ .

### C. Algorithm

An algorithm for the method in Fig. 1 is given in Fig. 4. Before we prove Theorem IV.1, we focus on the two operations in Fig. 4 that determine the complexity of the algorithm: The initialization of the data structures and the execution of line 20 which is part of the computation of the sets  $T$  and  $T_0$ .

1) *Initialization of the data structures:* The arrays  $z$  and  $i_z$  can be obtained in time linear in  $n+r+\nu$  by transposing  $\mathcal{X}$  when  $\mathcal{X}$  is available in the CCS-format [17, Section 2]. Roughly speaking, the main part of the transposing algorithm in [18] consists of a loop over all indices  $k \in [1; \nu]$ . In the body of the loop,  $z$  is computed as follows. If  $k$  is an index of the column  $j$  one sets  $z(l) := j$  for a suitable index  $l$  of the row  $s(k)$ . Additionally, one may initialize  $\tilde{s}$  by setting  $\tilde{s}(k) := l$  in the same loop. Thus,  $\tilde{s}$  clearly satisfies (3). Similarly,  $\tilde{z}$  is then obtained by transposing  $\mathcal{X}^T$ . Hence, the arrays  $z, i_z, \tilde{z}, \tilde{s}$  are initialized with complexity  $\mathcal{O}(n+r+\nu)$ .

**Input**  $s, \tilde{s}, i_s, z, \tilde{z}, i_z, c, w, j$   
**Require:**  $\mathcal{X}_{w,j} = *$  and  $l \in [i_z(w); i_z(w+1)[$  such that  
 $z(l) = j$ .  
1:  $\tilde{j} := \tilde{z}(l)$  (Note that  $s(\tilde{j}) = w$ )  
2:  $k := \tilde{s}(i_s(j) + c(j) - 1)$   
3: **if**  $c(j) > 1$  **then**  
4: Swap entries at positions  $\tilde{j}$  and  $i_s(j) + c(j) - 1$  in each of the arrays  $s$  and  $\tilde{s}$ .  
5:  $\tilde{z}(l) := i_s(j) + c(j) - 1$   
6:  $\tilde{z}(k) := \tilde{j}$   
7: **end if**  
8: **if**  $c(j) > 0$  **then**  
9: Decrement  $c(j)$ .  
10: **end if**  
**Output**  $s, \tilde{s}, \tilde{z}, c$

Fig. 3. Procedure to remove  $w$  from the representation of  $V \cap \text{NZR}(j)$

It is not hard to see that the initialization of the array  $c$  and the data structures for  $V$ ,  $T$  and  $T_0$  has time complexity  $\mathcal{O}(n+r)$ .

2) *Computation of  $T$  and  $T_0$ :* The computation of  $T$  and  $T_0$  in lines 3–7 in Fig. 1 is implemented by iteratively updating the representation of  $T$  and  $T_0$ . The update consists of two operations:

- Removing a row index  $w$  (due to line 17 in Fig. 1) from the representation of the sets  $V \cap \text{NZR}(j)$  for all  $j$ , and
- inserting or removing  $j$  from  $T$  ( $T_0$ , respectively) depending on the new value of  $c(j) = |V \cap \text{NZR}(j)|$ .

The algorithm for the first operation is given in Fig. 3: The array  $s$  is updated in line 4 to satisfy (7). Consequently, an update of  $\tilde{s}$  and  $\tilde{z}$  is necessary (lines 4, 5 and 6). The correctness of the updates in Fig. 3 is formalized in Lemma IV.7 below. Prior to that, we illustrate the iterative computation of  $T$  and the use of  $z, i_z, \tilde{s}$  and  $\tilde{z}$  by an example.

**IV.6 Example.** Let us perform the first steps in the overall algorithm in Fig. 4 for the pair  $(\mathcal{A}, \mathcal{B})$  given by (2) and for  $L = 0$ .

At line 8, the arrays  $s, z, i_s, i_z$  are given as in Fig. 2. The array  $c$  and the sets  $V$  and  $T$  are given as follows:

$$\begin{aligned} c &= (2, 1, 0, 2, 0, 1, 2, 1), \\ V &= \{1, 2, 3, 4, 5, 6\}, \\ T &= \{2, 6, 8\}. \end{aligned}$$

Since  $T \neq \emptyset$  we may pick 2 from  $T$  in line 13 to obtain  $w = s(i_s(2)) = 2$ . Therefore, for  $k \in [i_z(2); i_z(3)[ = \{2, 3\}$  we have to remove  $w = 2$  from the sets  $V \cap \text{NZR}(z(k))$  as required in line 20. Note that  $z(k) = 2, 7$  for  $k = 2, 3$ , i.e., the  $*$ -entries in row 2 are precisely in columns 2 and 7.

Updating  $V \cap \text{NZR}(2)$  and  $V \cap \text{NZR}(7)$ , respectively, in constant time requires the array  $\tilde{z}$  for the following reason. We need to access in constant time those positions  $\tilde{j}$  for which  $s(\tilde{j}) = w = 2$ . These indices are required since we need to swap the entries at positions  $\tilde{j}$  in  $s$  to possibly different positions due to the required property (7) of  $s$ . In order to avoid a search operation,  $\tilde{z}$  is introduced.  $\tilde{z}$  provides

**Input**  $s, i_s$   
**Require:**  $L \in \{0, 1\}$   
1: Transpose  $\mathcal{X}$ , initialize  $c$  and set  $V := [1; n]$ .  
2: **if**  $L = 0$  **then**  
3:  $T := \{v \in [1; n+r] \mid c(v) = 1\}$   
4: **else**  
5:  $T := \{v \in [n+1; n+r] \mid c(v) = 1\}$   
6:  $T_0 := \{v \in [1; n] \mid c(v) = 0\}$   
7: **end if**  
8: **while**  $V \neq \emptyset$  **do**  
9: **if**  $L = 0$  **or**  $T_0 = \emptyset$  **then**  
10: **if**  $T = \emptyset$  **then**  
11: **break**  
12: **end if**  
13: Pick  $v \in T$ .  
14:  $\{w\} := V \cap \text{NZR}(v)$  (Note that  $w$  is unique.)  
15: **else**  
16: Pick  $w \in T_0$ .  
17: **end if**  
18: **for all**  $k \in [i_z(w); i_z(w+1)[$  **do**  
19:  $j := z(k)$   
20: Execute the algorithm in Fig. 3 for  $w$  and  $j$ .  
21: **if**  $c(j) = 0$  **then**  
22:  $T := T \setminus \{j\}$   
23: **if**  $L = 1$  **and**  $j \in V$  **then**  
24:  $T_0 := T_0 \cup \{j\}$   
25: **end if**  
26: **else if**  $c(j) = 1$  **then**  
27: **if**  $L = 0$  **or**  $j \notin V$  **then**  
28:  $T := T \cup \{j\}$   
29: **end if**  
30: **end if**  
31: **end for**  
32: **if**  $L = 1$  **then**  
33: **if**  $c(w) = 1$  **then**  
34:  $T := T \cup \{w\}$   
35: **else if**  $c(w) = 0$  **then**  
36:  $T_0 := T_0 \setminus \{w\}$   
37: **end if**  
38: **end if**  
39:  $V := V \setminus \{w\}$ .  
40: **end while**  
**Output**  $V$

Fig. 4. Algorithm for the method given in Fig. 1

the required positions  $\tilde{j} = 3$  and  $\tilde{j} = 7$  as follows:  $\tilde{j} = \tilde{z}(k) = 3, 7$  for  $k = 2, 3$ , so  $s(3) = s(7) = 2$ .

Since the entry for column 2 in the array  $c$  equals 1, i.e.  $c(2) = 1$ , no swap is necessary for updating  $V \cap \text{NZR}(2)$ . The update is finished by decrementing  $c(2)$  to 0. Therefore, in line 22, we remove 2 from  $T$  to temporally obtain  $T = \{6, 8\}$ . In the representation of  $V \cap \text{NZR}(7)$ , a swap is necessary as  $c(7) = 2$ . So for  $\tilde{j} = 7$  (note that we identified  $\tilde{j}$  previously), positions  $\tilde{j}$  and  $i_s(7) + c(7) - 1 = 8$  in  $s$  are swapped, and  $c(7)$  is decremented to equal 1. Hence, 7 is inserted to  $T$  in line 28. Thus, we have  $T = \{6, 7, 8\}$  and

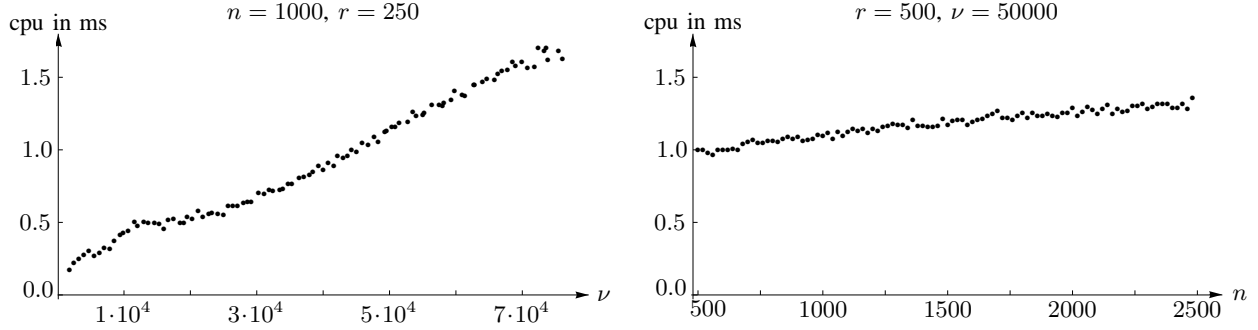


Fig. 5. Run time to verify strong structural controllability for  $\lambda = 0$  in dependence of  $\nu$  and  $n$  for randomly chosen pairs of structural matrices  $(\mathcal{A}, \mathcal{B}) \in \{0, *\}^{n \times (n+r)}$  such that  $(\mathcal{A}, \mathcal{B})$  is strong structurally controllable for  $\lambda = 0$ .  $\nu$  denotes the number of  $*$ -entries in  $(\mathcal{A}, \mathcal{B})$ . The run time to test strong structural controllability for  $\lambda \neq 0$  is within a factor 1.1 of the run time for  $\lambda = 0$ . The underlying implementation of the algorithm in Fig. 4 was executed on a Intel Core CPU i7-3770S (3.10 GHz).

$V = \{1, 3, 4, 5, 6\}$  at line 39.

As  $s$  has changed,  $\tilde{z}$  and  $\tilde{s}$  need to be updated accordingly by means of swapping entries. The access to the required positions is indicated in Fig. 2 and is similar to the access operations as detailed in this example. The updated arrays are as follows ( $z$  remains unchanged, the underlined entries below are those that have changed):

$$\begin{aligned} c &= (2, \underline{0}, 0, 2, 0, 1, \underline{1}, 1), \\ s &= (3, 5, 2, 1, 6, 4, \underline{3}, \underline{2}, 6), \\ \tilde{s} &= (4, 7, 2, 1, 8, 6, \underline{5}, \underline{3}, 9), \\ \tilde{z} &= (4, 3, \underline{8}, 1, \underline{7}, 6, 2, 5, 9). \end{aligned}$$

□

**IV.7 Lemma.** Let  $\mathcal{X}_{w,j} = *$  and let  $l$  be an index of the row  $w$  such that  $z(l) = j$ . Suppose that all inputs in Fig. 3 satisfy their defining properties for  $V \subseteq [1; n]$ . After the termination of the algorithm,  $s$  and  $c$  satisfy (7) for  $V \setminus \{w\}$  in place of  $V$ . Moreover, the arrays  $\tilde{s}$  and  $\tilde{z}$  satisfy (3) and (4), respectively.

*Proof:* The only assertion that requires a proof is the correct update of  $\tilde{z}$  after having changed  $s$  in line 4. Denote by  $\tilde{s}_0$  and  $\tilde{z}_0$  the arrays  $\tilde{s}$  and  $\tilde{z}$  prior to the execution of the algorithm. Inductively, we may assume that  $\tilde{s}_0$  and  $\tilde{z}_0$  satisfy (3) and (4) in place of  $\tilde{s}$  and  $\tilde{z}$ , respectively. By Lemma IV.5 we need to verify (5) for those positions  $k$  of  $\tilde{z}$  and  $\tilde{s}$  whose entries have changed after line 6. Indeed, (5) is valid as

$$\begin{aligned} \tilde{s}(\tilde{z}(l)) &= \tilde{s}_0(\tilde{j}) = \tilde{s}_0(\tilde{z}_0(l)) = l, \\ \tilde{s}(\tilde{z}(k)) &= \tilde{s}(\tilde{j}) = \tilde{s}_0(i_s(j) + c(j) - 1) = k, \\ \tilde{z}(\tilde{s}(\tilde{j})) &= \tilde{z}(\tilde{s}_0(i_s(j) + c(j) - 1)) = \tilde{z}(k) = \tilde{j}, \\ \tilde{z}(\tilde{s}(i_s(j) + c(j) - 1)) &= \tilde{z}(\tilde{s}_0(\tilde{j})) = \tilde{z}(l) = i_s(j) + c(j) - 1, \end{aligned}$$

hence the proof is finished. □

Due to the arrays  $\tilde{s}$  and  $\tilde{z}$ , the representation of the sets  $V \cap \text{NZR}(\cdot)$  can be updated in constant time, which is the key ingredient to Theorem IV.1 as we will see in the following subsection.

3) *Correctness and complexity:* Lines 8–16 of Fig. 1 clearly correspond to lines 9–17 in the algorithm in Fig. 4. The computation of  $T$  in lines 4 and 6 in Fig. 1 is realized in lines 18–38 in Fig. 4 as discussed above. We remark that lines 32–38 in Fig. 4 are required as the set  $V$  contains  $w$  during the execution of lines 23 and 27 in contrast to line 6 in Fig. 1. It follows that the outputs of the algorithms in Fig. 1 and Fig. 4 coincide.

The complexity of the algorithm in Fig. 4 proves Theorem IV.1. Indeed, lines 1–7 are executed in time linear in  $n+r+\nu$  (see Subsection IV-C.1). The while loop in line 8 terminates after at most  $n$  iterations (see Fig. 1). Together with the for loop in line 18, it yields an overall complexity of  $\mathcal{O}(n + \nu)$  for the while loop. Finally, the output of  $V$  requires time linear in  $n$ , so that Theorem IV.1 is proved.

## V. COMPUTATIONAL RESULTS

In this section, we analyze the performance of the implementation of the algorithm in Fig. 4 in two aspects: Linearity and the application of the algorithm to the minimization problem as discussed in Section I. The programming language in which the algorithm is implemented is C including Fortran routines of [18]. The computations were executed on a Intel Core CPU i7-3770S (3.10 GHz).

### A. Linearity

The linearity in  $\nu$  and  $n$  of the algorithm in Fig. 4 for  $L = 0$  is illustrated in Fig. 5. The matrices for which the computational time was recorded are chosen at random such that each is strong structurally controllable for  $\lambda = 0$ . The run time for  $L = 1$  and matrices that are strong structurally controllable for every  $\lambda \neq 0$  is within a factor 1.1 of the run time for  $L = 0$  but it is not illustrated in Fig. 5.

### B. Minimization

The matrix  $\mathcal{B}$  given in Example IV.2 is indeed one with the minimum number of columns such that the pair  $(\mathcal{A}, \mathcal{B})$  as defined in (2) is strong structurally controllable. This is verified in 0.56 milliseconds of cpu time by testing all possible candidates  $\mathcal{B}$ . We emphasize that the number of columns required in this case is strictly less than 3. In

contrast, the minimum number of columns obtained by the minimization algorithm presented in [11] is 3 due to restricting  $\mathcal{B}$  to have precisely one  $*$ -entry per column. As detailed in the introduction, real applications benefit from a reduced number of columns required for  $\mathcal{B}$  such that  $(\mathcal{A}, \mathcal{B})$  is strong structurally controllable.

The investigation of structural properties of electrical networks is quite popular, e.g. [11], [19]–[21]. In [11, Section IV, p. 418], a 5-bus power system is given in terms of a structural matrix  $\mathcal{A} \in \{0, *\}^{16 \times 16}$ . An application of our algorithm in Fig. 4 to find a structural matrix  $\mathcal{B}$  with a minimum number of columns such that  $(\mathcal{A}, \mathcal{B})$  is strong structurally controllable results in a structural matrix  $\mathcal{B}$  having 3 columns. It takes 437 seconds for this task using a parallel computation on 6 threads.

## REFERENCES

- [1] E. D. Sontag, *Mathematical control theory*, 2nd ed., ser. Texts in Applied Mathematics. New York: Springer-Verlag, 1998, vol. 6, deterministic finite-dimensional systems.
- [2] H. Mayeda and T. Yamada, “Strong structural controllability,” *SIAM J. Control Optim.*, vol. 17, no. 1, pp. 123–138, 1979.
- [3] K. J. Reinschke, F. Svaricek, and H.-D. Wend, “On strong structural controllability of linear systems,” in *Proc. 31st IEEE Conf. Decision and Control (CDC), Tucson, Arizona, USA.* IEEE, Dec. 1992, pp. 203–208.
- [4] J. C. Jarczyk, F. Svaricek, and B. Alt, “Strong structural controllability of linear systems revisited,” in *Proc. 50th IEEE Conf. Decision and Control (CDC) and European Control Conference (ECC), Orlando, FL, U.S.A., 12–15 Dec. 2011.* New York: IEEE, 2011, pp. 1213–1218.
- [5] A. Chapman and M. Mesbahi, “On strong structural controllability of networked systems: A constrained matching approach,” in *American Control Conference (ACC), 2013, June 2013*, pp. 6126–6131.
- [6] C. Hartung, G. Reißig, and F. Svaricek, “Characterization of strong structural controllability of uncertain linear time-varying discrete-time systems,” in *Proc. 51st IEEE Conf. Decision and Control (CDC), Maui, Hawaii, U.S.A., 10–13 Dec. 2012.* New York: IEEE, 2012, pp. 2189–2194. [Online]. Available: <http://dx.doi.org/10.1109/CDC.2012.6426326>
- [7] C. Hartung, G. Reißig, and F. Svaricek, “Sufficient conditions for strong structural controllability of uncertain linear time-varying systems,” in *Proc. 2013 American Control Conference (ACC), Washington, DC, U.S.A., 17–19 June 2013*, 2013, pp. 5895–5900.
- [8] C. Hartung, G. Reißig, and F. Svaricek, “Necessary Conditions for Structural and Strong Structural Controllability of Linear Time-Varying Systems,” in *Proc. European Control Conference (ECC), Zürich, Switzerland, 17–19 Jul. 2013*, 2013, pp. 1335–1340.
- [9] G. Reißig, C. Hartung, and F. Svaricek, “Strong structural controllability and observability of linear time-varying systems,” 2014, [arXiv:1306.1486](https://arxiv.org/abs/1306.1486). [Online]. Available: <http://dx.doi.org/10.1109/TAC.2014.2320297>
- [10] Y.-Y. Liu, J.-J. Slotine, and A.-L. Barabasi, “Controllability of complex networks,” *Nature*, vol. 473, no. 7346, pp. 167–173, May 2011.
- [11] S. Pequito, N. Popli, S. Kar, M. Ilic, and A. Aguiar, “A framework for actuator placement in large scale power systems: Minimal strong structural controllability,” in *Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 2013 IEEE 5th International Workshop on*, Dec 2013, pp. 416–419.
- [12] Y.-Y. Liu, J.-J. Slotine, and A.-L. Barabasi, “Control centrality and hierarchical structure in complex networks,” *PLoS ONE*, vol. 7, no. 9, p. e44459, 09 2012. [Online]. Available: <http://dx.doi.org/10.1371/journal.pone.0044459>
- [13] C. Commault and J.-M. Dion, “Input addition and leader selection for the controllability of graph-based systems,” *Automatica*, vol. 49, no. 11, pp. 3322 – 3328, 2013. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0005109813003695>
- [14] R. Barrett, M. Berry, T. F. Chan, and et al., *Templates for the solution of linear systems: building blocks for iterative methods*. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 1994. [Online]. Available: <http://dx.doi.org/10.1137/1.9781611971538>
- [15] G. Reißig, “Local fill reduction techniques for sparse symmetric linear systems,” *Arch. Elektrotech.*, vol. 89, no. 8, pp. 639–652, Sep. 2007. [Online]. Available: <http://dx.doi.org/10.1007/s00202-006-0042-2>
- [16] G. Reißig, “Fill reduction techniques for circuit simulation,” *Arch. Elektrotech.*, vol. 90, no. 2, pp. 143–146, Dec. 2007. [Online]. Available: <http://dx.doi.org/10.1007/s00202-007-0061-7>
- [17] F. G. Gustavson, “Two fast algorithms for sparse matrices: Multiplication and permuted transposition,” *ACM Trans. Math. Softw.*, vol. 4, no. 3, pp. 250–269, Sep. 1978. [Online]. Available: <http://doi.acm.org/10.1145/355791.355796>
- [18] Y. Saad, “Sparskit: a basic tool kit for sparse matrix computations - version 2,” 1994.
- [19] G. Reißig and U. Feldmann, “Computing the generic index of the circuit equations of linear active networks,” in *Proc. 1996 IEEE Int. Symp. on Circuits and Systems (ISCAS), Atlanta, GA, U.S.A., May 12–15*, vol. III, 1996, pp. 190–193. [Online]. Available: <http://dx.doi.org/10.1109/ISCAS.1996.541512>
- [20] G. Reißig, “Extension of the normal tree method,” *Internat. J. Circuit Theory Appl.*, vol. 27, no. 2, pp. 241–265, 1999, <http://www.reiszig.de/gunther/pubs/1999ijcta.abs.html>, Erratum in vol. 28, no. 1, 2000, p. 99. [Online].
- [21] G. Reißig and U. Feldmann, “A simple and general method for detecting structural inconsistencies in large electrical networks,” *IEEE Trans. Circuits Systems I Fund. Theory Appl.*, vol. 50, no. 11, pp. 1482–1485, Nov. 2003, author’s file: <http://www.reiszig.de/gunther/pubs/i03diagnosis.abs.html>. [Online]. Available: <http://dx.doi.org/10.1109/TCSI.2003.818620>